Artificial Intelligence Lab
AL-2002

Lab 02

Instructor: Hurmat Hidayat

Semester: Spring 2023

Artificial Intelligence Lab 02

Objectives

The objective of this lab is to understand AI agents and their usage, describe different types of AI agents and develop agents using python, including simple reflex, model-based, goal-based, and utility-based agents.

Learning Outcomes

1. To understand the concept of percept, state, and rule in AI agents and how they are used to make decisions.
2. To develop a real-world scenario where an AI agent can be implemented and show how it can improve efficiency and productivity.

Table of Contents

Agents in Artificial Intelligence

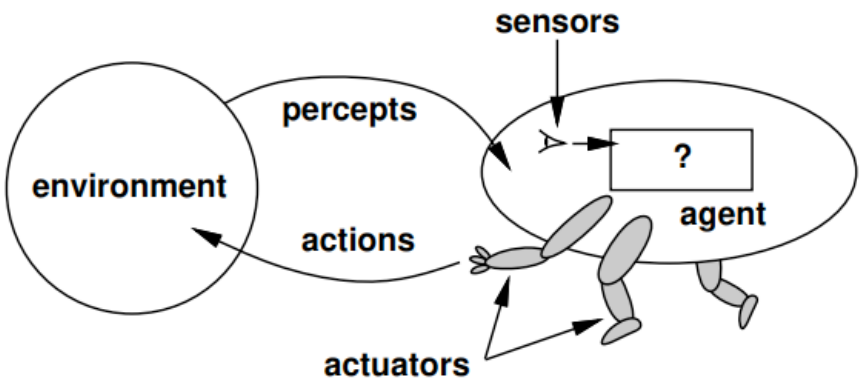
A rational agent could be anything that makes decisions, as a person, firm, machine, or software. It carries out an action with the best outcome after considering past and current percepts(agent's perceptual inputs at a given instance). Agents include :

- Animal agents
- Human agents
- Robotic agents (robots)
- Software agents (softbots)
    - internet agents, crawler, webbot, email agent, search agent, etc.

Agents and Environments

An AI system is composed of an agent and its environment. The agents act in their environment. The environment may contain other agents. An agent is anything that can be viewed as :

- perceiving its environment through sensors and
- acting upon that environment through actuators



Sensors and Actuators

A **sensor** measures some aspect of the environment in a form that can be used as input by an agent

- vision, hearing, touch
- radio, infrared, GPS, wireless signals

- active sensing: send out a signal (such as radar or ultrasound) and sense the reflection of this signal off of the environment i.e. IoT (Internet of Things)

Perception provides agents with information about the world they inhabit by interpreting the response of sensors.

Actuators

- hands, legs, vocal tract etc
- automated taxi: those available to a human driver e.g., accelerator, steering, braking and so on

Agent Terminology

- **Performance Measure of Agent** − It is the criteria, which determines how successful an agent is.
- **Behavior of Agent** − It is the action that agent performs after any given sequence of percepts.
- **Percept** − It is agent's perceptual inputs at a given instance.
- **Percept Sequence** − It is the history of all that an agent has perceived till date.
- **Agent Function** − It is a map from the precept sequence to an action.

Structure of Intelligent Agent

To understand the structure of Intelligent Agents, we should be familiar with Architecture and Agent programs. Architecture is the machinery that the agent executes on. It is a device with sensors and actuators, for example, a robotic car, a camera, a PC. Agent program is an implementation of an agent function. An agent function is a map from the percept sequence(history of all that an agent has perceived to date) to an action.

Agent = Architecture + Agent Program

Agent Programs

An agent program is a computer program that is designed to autonomously carry out tasks on behalf of its user or another program. Agent programs are often used in artificial intelligence (AI) and multi-agent systems, and are designed to be able to perceive, reason, and act in order to accomplish their goals.

Rationality

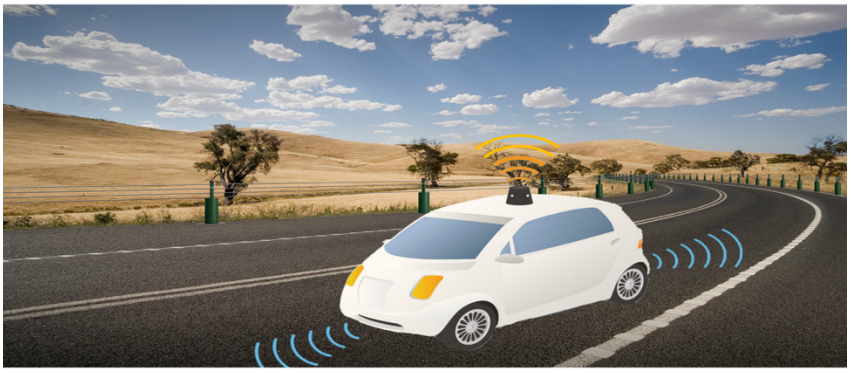
Rationality is nothing but status of being reasonable, sensible, and having good sense of judgment. Rationality is concerned with expected actions and results depending upon what the agent has perceived. Performing actions with the aim of obtaining useful information is an important part of rationality.

PEAS (Performance/Environment/Actuators/Sensors)

PEAS is a type of model on which an AI agent works upon. When we define an AI agent or rational agent, then we can group its properties under PEAS representation model. It is made up of four words:

- **P:** Performance measure
- **E:** Environment
- **A:** Actuators
- **S:** Sensors

PEAS for self-driving cars



Let's suppose a self-driving car then PEAS representation will be:

- Performance: Safety, time, legal drive, comfort
- Environment: Roads, other vehicles, road signs, pedestrian
- Actuators: Steering, accelerator, brake, signal, horn
- Sensors: Camera, GPS, speedometer, odometer, accelerometer, sonar.

Example of Agents with their PEAS representation

Agent	Performance measure	Environment	Actuators	Sensors
1. Medical Diagnose	Healthy patient, Minimized cost	Patient, Hospital, Staff	Tests, Treatments	Keyboard (Entry of symptoms)
2. Vacuum Cleaner	Cleanness, Efficiency, Battery life, Security	Room, Table, Wood floor, Carpet, Various obstacles	Wheels, Brushes, Vacuum Extractor	Camera, Dirt detection sensor, Cliff sensor, Bump Sensor, Infrared Wall Sensor
3. Part - picking Robot	Percentage of parts in correct bins	Conveyor belt with parts, Bins	Jointed Arms, Hand	Camera, Joint angle sensors

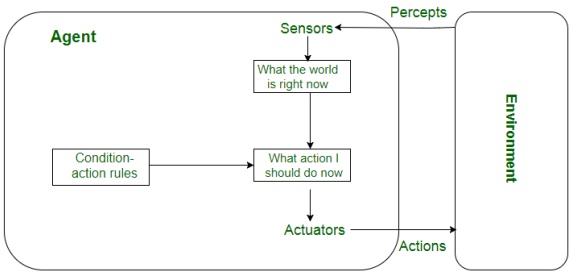
Types of Agents

Agents can be grouped into five classes based on their degree of perceived intelligence and capability :

1. Simple Reflex Agents
2. Model-Based Reflex Agents
3. Goal-Based Agents
4. Utility-Based Agents
5. Learning Agent

Simple Reflex Agents

Simple reflex agents ignore the rest of the percept history and act only on the basis of the **current percept**. Percept history is the history of all that an agent has perceived to date. The agent function is based on the **condition-action rule**. A condition-action rule is a rule that maps a state i.e, condition to an action. If the condition is true, then the action is taken, else not. This agent function only succeeds when the environment is fully observable. For simple reflex agents operating in partially observable environments, infinite loops are often unavoidable. It may be possible to escape from infinite loops if the agent can randomize its actions.




```
def SIMPLE-REFLEX-AGENT(percept)
persistent: rules, a set of condition-action rules
    state ← INTERPRET-INPUT(percept)
    rule ← RULE-MATCH(state, rules)
    action ← rule.ACTION
    return action
```

The Sorting Robot: Organizing a Warehouse with Simple Reflex Agent Program

Imagine a robot that is placed in a warehouse to sort and organize boxes. The boxes come in different colors, red, blue, and green. The robot's task is to sort the boxes and place them in the designated areas for each color.

The robot's **percepts** in this scenario are the visual inputs it receives from its camera, and the **states** are the boxes it sees in the warehouse (red, blue, and green). The **robot's rules** are simple:

- if it sees a red box, place it in the red area,
- if it sees a blue box, place it in the blue area, and
- if it sees a green box, place it in the green area.

Here is the robot's simple reflex agent program

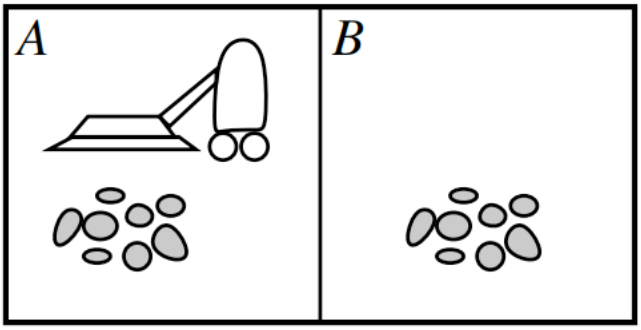
```
percept = ['red', 'blue','green']
state = ['red_box', 'blue_box','green_box']
rules = ['place in red area', 'place in blue area','place in green area']

def getState(percept_value):
    index = -1
    for x in percept:
        index = index+1
        if x == percept_value:
            return state[index]

def getRules(state_value):
    index = -1
    for i in state:
        index = index+1
        if i == state_value:
            return rules[index]

def simpleReflexAgent(percep):
    return getRules(getState(percep))

#set a variable to keep track of the number of boxes
box_count = 0
while box_count < 10:
    # robot receive visual input from its camera
    visual_input = input("What color of box do you see?")
    rule = simpleReflexAgent(visual_input)
    print(rule)
    # robot execute the rule
    box_count += 1
    # increment the box_count variable after each iteration
```

Problems with Simple reflex agents

Problems with Simple reflex agents are :

- Very limited intelligence.
- No knowledge of non-perceptual parts of the state.
- Usually too big to generate and store.
- If there occurs any change in the environment, then the collection of rules need to be updated.

A Vacuum-cleaner Agent



This particular world has just two locations: squares A and B. The vacuum agent perceives which square it is in and whether there is dirt in the square. It can choose to move left, move right, suck up the dirt, or do nothing. One very simple agent function is the following:

if the current square is dirty, then suck, otherwise move to the other square.

A simple program for the agent function of vacuum-world is shown below:

- **Percepts:** Location and status, e.g., [A,Dirty]
- **Actions:** Left, Right, Suck, NoOp

function Vacuum-Agent([location,status]) returns an action

- *if* status = Dirty *then* return Suck
- *else if* location = A *then* return Right
- *else if* location = B *then* return Left

Lab Tasks

Task 1.

Imagine a single-floor office building with a fire alarm system that is controlled by a simple reflex agent. The system has smoke detectors and temperature sensors placed throughout the building to detect any signs of fire.

The agent's rules are as follows:

- if smoke is detected, the alarm will sound and the sprinkler system will activate to put out the fire;
- if a high temperature is detected, the alarm will sound and the fire department will be called.
- If neither smoke nor high temperature are detected, the system remains in its normal state with the alarm off and the sprinkler system deactivated.

The goal of the agent is to keep the building and its occupants safe by quickly and efficiently responding to any signs of fire. Write a program to develop a simple reflex Agent.

Task 2.

An Automatic Watering System is set up in a greenhouse. The system has sensors that detect the moisture level in the soil, and a control unit that operates the watering system. The task of the simple reflex agent program is to control the watering system based on the moisture level of the soil.

Percepts: Moisture level sensor

States: Dry soil, Moist soil, Wet soil

Rules:

1. If the moisture level sensor detects dry soil, the agent activates the watering system to water the plants.
2. If the moisture level sensor detects moist soil, the agent keeps the watering system off to avoid over watering the plants.
3. If the moisture level sensor detects wet soil, the agent deactivates the watering system to prevent waterlogging.

Note: Implement Automatic Watering System using OOP approach with error handling and logging. Make the system flexible by adding threshold values for each state.

References

1. https://www.simplilearn.com/what-is-intelligent-agent-in-ai-types-function-article
2. https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_agents_and_environments.htm
3. https://www.geeksforgeeks.org/agents-artificial-intelligence/
4. https://www.studocu.com/row/document/riphah-international-university/computer-sciences/cs6-1-lab-06-good/39006543